

AF #
2151

PATENT
Customer No. 22,852
Attorney Docket No. 06502.0269

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:)
)
Peter C. JONES et al.) Group Art Unit: 2151
)
Application No.: 09/332,029) Examiner: T. Ho
)
Filed: June 14, 1999)
)
For: METHOD AND SYSTEM FOR)
DYNAMIC PROXY CLASSES)

RECEIVED

JAN 08 2004

Technology Center 2100

Mail Stop Appeal Brief--Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

TRANSMITTAL OF SUPPLEMENTAL APPEAL BRIEF PURSUANT TO
37 C.F.R. 1.193(b)(ii)

Transmitted herewith in triplicate is the SUPPLEMENTAL APPEAL BRIEF in this application in response to the Office Action filed October 7, 2003. Appellants request that the fees paid for the Notice of Appeal filed May 19, 2003, and Appeal Brief filed July 14, 2003, be applied to this Supplemental Appeal Brief (see M.P.E.P. § 1208.02).

This application is on behalf of

☐ Small Entity ☒ Large Entity

Pursuant to 37 C.F.R. 1.17(c), the fee for filing the Supplemental Appeal Brief is:

☐ \$160.00 (Small Entity)

☒ \$330.00 (Large Entity)

FINNEGAN
HENDERSON
FARABOW
GARRETT &
DUNNER LLP

1300 I Street, NW
Washington, DC 20005
202.408.4000
Fax 202.408.4400
www.finnegan.com



- ☒ As explained, Appellants request that the fee of \$320.00 previously paid for the Appeal Brief filed July 14, 2003, be applied to this Supplemental Appeal Brief.¹

PETITION FOR EXTENSION. If any extension of time is necessary for the filing of this Supplemental Appeal Brief, and such extension has not otherwise been requested, such an extension is hereby requested, and the Commissioner is authorized to charge necessary fees for such an extension or for any other appropriate fees for filing this Supplemental Appeal Brief to our Deposit Account No. 06-0916. A duplicate copy of this paper is enclosed for use in charging the deposit account.

FINNEGAN, HENDERSON, FARABOW,
GARRETT & DUNNER, L.L.P.

Dated: January 6, 2004

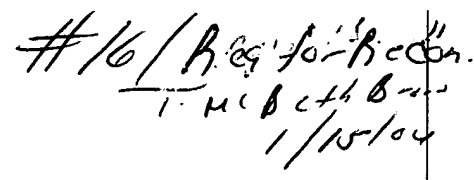
By: 

Joseph E. Palys
Reg. No. 46,508

FINNEGAN
HENDERSON
FARABOW
GARRETT &
DUNNER LLP

1300 I Street, NW
Washington, DC 20005
202.408.4000
Fax 202.408.4400
www.finnegan.com

¹ The fees required under 37 C.F.R. §§1.17(b) and (c) have increased from \$320.00 to \$330.00 on October 1, 2003. Because Appellants are entitled to reinstate the Appeal Brief filed on July 14, 2003, Appellants request that the difference between the fees be waived. Should, however, these fees not be waived, Appellants authorize the Commissioner to charge any remaining balance to our Deposit Account No. 06-0916.



Customer Number 22,852
Attorney Docket No. 06502.0269

In re Application of:

Peter C. JONES et al.

Group Art Unit: 2151

Serial No.: 09/332,029

Examiner: T. Ho

Filed: June 14, 1999

For: METHOD AND SYSTEM FOR
DYNAMIC PROXY CLASSES

RECEIVED

JAN 08 2004

Technology Center 2100

Mail Stop Appeal Brief--Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

SUPPLEMENTAL APPEAL BRIEF UNDER 37 C.F.R. §§ 1.193(b)(ii) and 1.192

In response to the Office Action dated October 7, 2003, and pursuant to 37 C.F.R. §§ 1.192 and 1.193(b)(ii), Appellants present in triplicate their Supplemental Appeal Brief. Because the Examiner reopened prosecution in response to Appellants' Appeal Brief filed July 14, 2003, Appellants request that the fees required under 37 C.F.R. §§ 1.17(b) and (c) that accompanied the Notice of Appeal filed May 19, 2003, and the Appeal Brief filed July 14, 2003, respectively, be applied to this Supplemental Appeal Brief.¹

31/07/2004 JABDD01

00000040 060916 09332029

FINNEGAN
HENDERSON
FARABOW
GARRETT &
DUNNER LLP

330.00 DA

¹ The fees required under 37 C.F.R. §§1.17(b) and (c) have increased from \$320.00 to \$330.00 on October 1, 2003. Because Appellants are entitled to reinstate the Appeal Brief filed on July 14, 2003, Appellants request that the difference between the fees be waived. Should, however, these fees not be waived, Appellants authorize the Commissioner to charge any remaining balance to our Deposit Account No. 06-0916.

1300 I Street, NW
Washington, DC 20005
202.408.4000
Fax 202.408.4400
www.finnegan.com

This is a supplemental appeal to the Board of Patent Appeals and Interferences from a decision by the Examiner reopening prosecution and rejecting claims 1, 2, 4-7, and 9-13 in response to the Appeal Brief filed by Appellants on July 14, 2003. The appealed claims are set forth in the Appendix. If additional fees are required, please charge the deficiencies to Deposit Account No. 06-0916. If a fee is required for an extension of time under 37 C.F.R. § 1.136 and such fee is not accounted for above, Appellants petition for such an extension and request that the fee be charged to Deposit Account No. 06-0916.

I. REAL PARTY IN INTEREST

The real party in interest is Sun Microsystems, Inc., a corporation of Delaware.

II. RELATED APPEALS AND INTERFERENCES

There are no known related pending appeals or interferences directly affected by or having a bearing on the decision in the pending appeal.

III. STATUS OF CLAIMS

Claims 1, 2, 4-7, and 9-13 were finally rejected on December 18, 2002.

Following a Request for Reconsideration filed on February 11, 2003, Appellants filed an Appeal Brief on July 14, 2003. In response, the Examiner has reopened prosecution and rejected claims 1, 2, 4-7, and 9-13. Accordingly, these claims are the subject of this supplemental appeal. The claims on appeal are set forth under the heading

APPENDIX. In the Office Action dated October 7, 2003, the Examiner rejected claims 1, 2, 4, 6, 7, 9, and 12 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. (U.S. Patent No. 5,805,885) in view of Ouellette (U.S. Patent No. 6,442,619); rejected claims 5 and 10 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette, and in further view of Hailpern et al. (U.S. Patent No. 6,257,937); and rejected claims 11 and 13 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette and Hailpern et al., and further in view of Hughes (U.S. Patent No. 6,345,382).

IV. STATUS OF AMENDMENTS

Appellants filed one amendment on October 7, 2002 canceling claims 3 and 8 and amending claims 1, 6, 12, and 13 as indicated in the attached Appendix.

V. SUMMARY OF INVENTION

Many computing systems today utilize programs written in object-oriented programming languages. In some object-oriented programming languages, such as the Java programming language provided by Sun Microsystems, Inc., classes are defined that are a template for creating objects. Classes contain data members that store data and methods that act upon the data. Additionally, such programming languages uses interfaces that are lists of methods. An interface differs from a class because it declares methods and is not capable of implementing them.

The source code for classes, interfaces, and methods are usually written by a

user and compiled on a computing system during a stage referred to as "compile-time."

After the source code is compiled, a computing system may execute the source code during a stage known as "runtime." Because a class typically has to be built offline and then compiled, the interfaces implemented by that class must be known while the class is being built before runtime. In some instances, however, a user or process may require use of an interface that may not be known or may be changing during runtime of a program. Conventionally, to implement a single functionality on multiple interfaces of varying types, code needs to be created for each interface to carry out the functionality. Further, a class needs to be generated before runtime that implements a list of interfaces, and code for each method of an interface being implemented needs to be written and compiled before runtime.

The present invention addresses the above problems by providing a system and method for dynamically generating a proxy class during runtime that implements a list of interfaces specified at runtime. Method invocations on an instance of the proxy class are encoded and dispatched to another object that handles method invocations. In one aspect related to the present invention, the proxy class generated at runtime does not require the implemented interfaces and their methods to be known before runtime. Accordingly, interfaces that may be used during execution of a program may change during runtime before the creation of the proxy class. Moreover, the proxy class may provide uniform implementation of multiple interface types without requiring specialized code for each interface.

VI. ISSUES

The issues in this Appeal are:

(1) whether the Examiner's rejection of claims 1, 2, 4, 6, 7, 9, and 12 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. and Ouellette can be affirmed when the references, alone or in combination, do not teach or suggest every recitation of these claims including, for example, generating at runtime a class that implements an interface specified at runtime, creating an instance of the class, dispatching a request to an object to facilitate processing of a method of the interface, and returning a result of the processed method by the object, as recited in claims 1, 6, and 12, and the specifying steps recited in claims 4 and 9;

(2) whether the Examiner's rejection of claims 5 and 10 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette, and further in view of Hailpern et al. can be affirmed when the references fail to teach or suggest every recitation of these claims including generating at runtime a class that implements an interface by generating code for each of the methods included in the interfaces and an invocation handler, as recited in these claims; and

(3) whether the Examiner's rejection of claims 11 and 13 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette and Hailpern et al., and further in view of Hughes can be affirmed when none of the references teaches or suggests every recitation of these claims including an invocation handler and proxy class, as recited in these claims.

VII. GROUPING OF CLAIMS

In the claims on appeal, claims 1, 5, 6, 10, 11, 12, and 13 are the independent claims. The claims on appeal do not stand or fall together. These claims should be considered in four groups:

Group I: 1, 2, 6, 7, and 12;

Group II: 4 and 9;

Group III: 5 and 10; and

Group IV: 11 and 13.

The claims have been placed in these groups due to their common subject matter. Appellants, however, have addressed the outstanding rejections in accordance with the rejections themselves instead of the above identified groupings.

VIII. ARGUMENT

a. Summary of Arguments

The arguments presented in this brief supplement those previously presented in the Appeal Brief filed on July 14, 2003. Because, however, the previous arguments are considered to be relevant to the above described issues, Appellants have explicitly incorporated them in this Supplemental Appeal Brief.

The Rejection of Claims 1, 2, 4, 6-9, and 12

Appellants appeal the rejection of claims 1, 2, 4, 6, 7, 9, and 12 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette because the references do not teach each and every step and/or element of these

claims. The Examiner's rejection should be reversed for the following reasons. First, the citations from Leach et al. relied upon by the Examiner do not show dispatching the request to an object, as recited in claims 1, 6, and 12. Second, the portions of Leach et al. cited by the Examiner relate to static aggregation that requires knowledge of interfaces before runtime, which teaches away from the recitations of claims 1, 6, and 12. Third, the dynamic aggregation aspects taught by Leach et al. merely require registering interfaces as they are instantiated, and do not process methods of the interfaces or dispatch a request to an object to facilitate such processing, as recited in claims 1, 6, and 12. Fourth, the Examiner's interpretation of Leach et al. in view of the combination of steps and/or elements recited in claims 1, 6, and 12, respectively, is improper. Fifth, Ouellette does not make up for the deficiencies of Leach et al. And sixth, there is no motivation or reasonable expectation of success in combining the teachings of Ouellette with those of Leach et al. to teach the process of returning a result of a processed method, as recited in claims 1, 6, and 12.

Further, the Examiner's rejection of claims 4 and 9 should be reversed because retrieving a reference to an aggregated interface is not the same as specifying an object to process method invocations on a created instance of a class that implements an interface specified at runtime.

The Rejection of Claims 5 and 10

Appellants appeal the rejection of claims 5 and 10 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette, and further in view of Hailpern

et al. because a prima facie case of obviousness has not been made by the Examiner. The rejection of claims 5 and 10 should be reversed because Leach et al., Ouellette, and Hailpern et al., alone or in combination, fail to teach or suggest the invocation handler recited in claims 5 and 10. Further, there is no reasonable expectation of success in implementing a virus scanner exception handler, as taught by Hailpern et al., in the method invocation processes taught by Leach et al. Moreover, the Examiner has failed to present any evidence to support the conclusion that these references, when combined, render claims 5 and 10 obvious. Further, presenting class definitions that are programmed before runtime, as described in table 3 of Leach et al., does not show generating a class at runtime that implements an interface by generating code for each of a plurality of methods of an interface indicated at runtime.

The Rejection of Claims 11 and 13

Appellants appeal the rejection of claims 11 and 13 under 35 U.S.C. § 103(a) as being unpatentable over Leach et al. in view of Ouellette and Hailpern et al., and further in view of Hughes because a prima facie case of obviousness has not been made by the Examiner. The rejection of claims 11 and 13 should be reversed because none of the applied references teaches or suggests dispatching a request to an invocation handler object and returning a value from the handler object to a proxy class instance, as recited in claim 11. Further, the references do not teach or suggest a processor for generating the proxy class at runtime, receiving a request to access a method of an interface specified at runtime, and dispatching the request to the object to facilitate

processing of the requested method of the interface, as recited in claim 13. In addition to the deficiencies noted above, the Examiner's assertion that Hughes teaches a proxy class that is generated at runtime is incorrect. Rather, the proxy class disclosed by Hughes includes code that is generated by a manufacturer before runtime.

b. Summary of Claims

Claim 1 is drawn to a method in a data processing system that comprises the steps of generating at runtime a class that implements an interface specified at runtime having a method and creating an instance of the class. The claimed process also involves receiving by the class instance a request to process the method of the interface, dispatching the request to an object to facilitate processing of the method of the interface, and returning a result of the processed method by the object.

Claim 6 is drawn to a computer-readable medium containing instructions for controlling a data processing system to perform the method described above with reference to claim 1. Claim 12 is drawn to a data processing system comprising elements that perform operations similar to the process steps described above with reference to claim 1. Claims 2, 4 and 7, 9 depend from claims 1 and 6, respectively, and are drawn to process steps for generating at runtime a class that implements more than one interface specified at runtime, and specifying an object to process method invocations on the instance, respectively.

Claim 5 is drawn to a method in a data processing system having an invocation handler that comprises the steps of receiving at runtime an indication of at least one

interface having a plurality of methods and generating at runtime a class that implements the interface by generating code for each of the methods that dispatches an invocation of the method to the invocation handler. Claim 10 is drawn to a computer-readable medium containing instructions for controlling a data processing system having an invocation handler to perform the method described above with reference to claim 5.

Claim 11 is drawn to a method in a data processing system having a proxy class implementing a list interfaces specified at runtime, an instance of the proxy class, and an invocation handler object for executing methods contained by the interfaces. The claimed method comprises the steps of determining interfaces to be implemented by the proxy class and generating at runtime the proxy class implementing the determined interfaces. The method further includes receiving a request by the proxy class instance to invoke a method of an interface implemented by the proxy class, dispatching the request to the invocation handler object, returning a value from the invocation handler object to the proxy class instance, and returning the value from the proxy class instance.

Claim 13 is drawn to a data processing system that comprises a memory and a processor. The memory contains a proxy class, an instance of the proxy class, an interface specified at runtime having methods, and an object for handling method invocations. The proxy class contained in the memory implements the interface. The processor generates the proxy class at runtime, receives a request to access a method

of the interface and dispatching the request to the object to facilitate processing of the requested method of the interface.

c. The Examiner's position that Leach et al. and Ouellette teach or suggest the recitations of claims 1, 2, 6, 7, and 12 is wrong because the references fail to disclose or suggest the steps and/or elements of dispatching the request and returning a result, as recited in the claims.

Appellants traverse the rejection of claims 1, 2, 6, 7, and 12 under 35 U.S.C. § 103(a) because a prima facie case of obviousness has not been made by the Examiner. To establish a prima facie case of obviousness under 35 U.S.C. § 103(a), each of three requirements must be met. First, the reference or references, taken alone or combined, must teach or suggest each and every element recited in the claims (see M.P.E.P. § 2143.03 (8th ed. 2001)). Second, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to combine the references in a manner resulting in the claimed invention. Third, a reasonable expectation of success must exist. Moreover, each of these requirements must "be found in the prior art, and not be based on applicant's disclosure" (see M.P.E.P. § 2143 (8th ed. 2001)).

In an attempt to reject claims 1, 2, 6, 7, and 12 under 35 U.S.C. § 103(a), the Examiner asserts that Leach et al. teaches every recitation of these claims except for the step of "returning a result of the processed method by the object." (See Office Action dated October 7, 2003, pages 3-4, paragraph 3). Appellants respectfully submit that the Examiner has failed to establish that Leach et al. teaches these recitations of

claims 1, 2, 6, 7, and 12. Further, the Examiner asserts that the request-response processes performed by Ouellette render these claims obvious in view of Leach et al. Appellants respectfully traverse the Examiner's position.

Leach et al. discloses a method and system for aggregating objects within an enclosed object. The method and system allows for static and dynamic aggregation. In static aggregation, an interface of an enclosing object knows in advance (e.g., before runtime) how to return an identifier to an external interface of an enclosed object. In dynamic aggregation, an enclosed object is added to the enclosing object after the enclosing object is instantiated. Once included in the enclosing object, a reference to an interface of the enclosed object may be returned in response to an external request to access the interface (see Leach et al., Abstract, col. 11, lines 23-34 and col. 21, line 65 to col. 22, line 21).

In contrast, claim 1 recites a combination of steps including, among other things, dispatching a request to an object to facilitate processing of a method of an interface specified at runtime, and returning a result of the processed method by the object.

The Examiner asserts that Leach et al. "discloses dispatching the request to an object to facilitate processing of the method of the interface (Fig. 7C, and requests received by an enclosed object are passed to the enclosing object, line 67 column 22 to line 1 column 23)" (see Office Action dated October 7, 2003, page 3, lines 10-12).

Appellants respectfully disagree with this assertion for several reasons.

First, the block diagram illustrated in Fig. 7C of Leach et al. (referenced by the Examiner) shows a multitype object ("MTO") after the addition of an interface using a

particular method (e.g., AddObject). A MTO is an object capable of aggregating objects of varying types (see Leach et al., col. 23, lines 14-16). The IUnknown interface included in the MTO is capable of providing a pointer to an external interface and/or other MTO interfaces (see Leach et al., col. 24, lines 27-44). Accordingly, the MTO taught by Leach et al. does not show dispatching a request to an object to facilitate processing of the method of an interface, as recited in claim 1.

Second, the ability of Leach et al. to pass requests received by an enclosed object to an enclosing object (cited by the Examiner, see Office Action dated October 7, 2003, page 3, lines 10-12) is associated with static aggregation (see Leach et al., col. 22, line 49 to col. 23, line 3). As explained, static aggregation requires that an enclosing object have advance (i.e., before runtime) knowledge of the interfaces it wishes to aggregate (see Leach et al., col. 11, lines 15-23). Claim 1 includes the step of generating at runtime a class that implements an interface specified at runtime having a method. Accordingly, the static aggregation process of passing of requests to an enclosing object discussed by Leach et al. cannot teach the dispatching step of claim 1 because this step includes dispatching the request to an object to facilitate processing of a method of the interface that is specified at runtime.

Third, the dynamic aggregation processes disclosed by Leach et al. in col. 23, lines 4-23, also fail to teach the dispatching step of claim 1. According to Leach et al., the enclosing object "provides a method for registering instantiated interfaces and for later retrieving references to them." (see Leach et al., col. 23, lines 5-7). Therefore, Leach et al. requires an enclosing object method to register interfaces as they are

instantiated and only provides references to the interfaces. Leach et al. does not teach processing the method of the interface, much less dispatching a request to an object to facilitate the processing of the method. Instead, Leach et al. teaches an enclosing object (e.g., multitype object) that uses a method to retrieve references to registered interfaces. Accordingly, Leach et al. cannot teach the dispatching step of claim 1 because the reference does not disclose dispatching a request to an object to facilitate processing of a method of an interface. The method invoked by the enclosing object as taught by Leach et al. is not an object. Further, the enclosing object taught by Leach et al. cannot be equivalent to the object of claim 1 that receives the dispatched request because the enclosing object performs aggregation services which are not associated with the processing of any methods of interfaces implemented by the enclosing object. The aggregation services merely add objects and interface lists to an enclosing object to allow the enclosing object to retrieve pointers to interfaces implemented by the enclosing object.

Additionally, Appellants submit that the Examiner has incorrectly interpreted Leach et al. in view of the combination of steps recited in claim 1. For example, the Examiner asserts that Leach et al. teaches generating at runtime a class that implements an interface specified at runtime having a method and creating an instance of the class (see Office Action, page 3, lines 4-7). In particular, the Examiner asserts that the "objects" disclosed in col. 11, lines 15 and 21-22 of Leach et al. are the same as the class instance created in claim 1. Appellants disagree. The objects created by Leach et al. are enclosed objects which are objects or interfaces added to an enclosing

object. The enclosed objects are those objects or interfaces implemented during runtime and are not the same as the class instance as specified in claim 1.

Also, as admitted by the Examiner, Leach et al. does not teach returning a result of the processed method by the object, as recited in claim 1. (See Office Action dated October 7, 2003, page 3, lines 13-14). To make up for this deficiency, the Examiner contends that the request-response processes performed by Quellette render this recitation obvious when combined with Leach et al. because "this allows the server object to process subsequent messages wherein specialized functionality does not appear in the server object, but is instead distributed at lower levels of the software hierarchy" (See Office Action dated October 7, 2003, page 3, lines 18-21). Appellants disagree.

Quellette describes a distributed computing system that uses a control program for processing request and response messages. The control program uses a common server object for receiving and processing these messages. In operation, the server object creates a request object for processing each request message by executing methods for performing an operation identified in a request message. As a result, the request object creates a response message. Further, the control program includes different types of objects for processing the request and response messages, such as a factory object, a mailbox object, a send mailbox object, and a message object (see Quellette, Abstract, and col. 2, line 15 to col. 3, line 40). Quellette processes a request message by creating an instance of the server object that orders a request object from a factory object class. The factory object creates a request object from a request object

class based on the type of operation requested. The request object adds itself to an active request list that registers and tracks active request objects awaiting responses from other objects. The factory object returns the request object to the server object, which invokes a method implemented by the request object for processing the requested operation. (see Ouellette, col. 6, lines 34 to col. 7, line 48).

Although Ouellette discloses a process where response messages are created based on corresponding request messages, the process is not compatible with the system disclosed by Leach et al. Accordingly, there is no motivation to combine the references or a reasonable expectation of success in the combination proposed by the Examiner.

The server object class described by Ouellette is not generated at runtime. Instead, the server object class is an abstract base class for any specialized server object class. (see Ouellette, col. 6, lines 27-33). Instances of the server object class has a pointer to the beginning of the active request list. (See Ouellette, col. 6, lines 11-18 and 34-36). Accordingly, there can be no motivation to combine the references or a reasonable expectation of success in implementing the server objects disclosed by Ouellette with the dynamic aggregation processes taught by Leach et al. The Examiner supports his conclusion that the proposed combination would be obvious because the server objects taught by Ouellette have a distributed software hierarchy (i.e., use of request objects, factory objects, and server objects for processing request messages). Appellants disagree. The use of class-subclass instances in the object-oriented framework disclosed by Ouellette does not suggest the use of a dynamically generated

class that implements an interface specified at runtime, as recited by claim 1. Indeed, the above assertion does not support a motivation for using a server object in combination, or in place of, the enclosing objects disclosed by Leach et al.

Furthermore, as explained, there can be no reasonable expectation of success in using a statically defined class (i.e., a server object) with Leach et al. because the reference uses dynamic aggregation techniques which include enclosing objects that have no compile time knowledge of enclosed objects or interfaces (see Leach et al., col. 11, lines 24-34).

Further, Leach et al. is not configured to return a result of a processed method. As admitted by the Examiner, the QueryInterface method implemented by Leach et al. returns pointers to exposed interfaces (see Final Office Action, page 3, lines 6-8). Returning pointers is not the same as returning a result of a processed method of the interface specified at runtime, as recited in claim 1. In fact, Leach et al. does not even disclose processing a method of an interface to return a result. Instead, Leach et al. processes a method in an enclosing object (e.g., multitype object) that returns references to interfaces added to the enclosing object (see Leach et al., col. 23, lines 5-7). Accordingly, Leach et al. cannot be compatible with a system that matches response messages with request messages, as taught by Ouellette because Leach et al. processes a single method of the enclosing object to return a pointer to an enclosed interface.

Because Leach et al. and Ouellette, alone or in combination, fail to teach or suggest every recitation of claim 1, Appellants respectfully request that the Board reverse the rejection of claim 1.

Claims 6 and 12 include recitations similar to those of claim 1. As explained, claim 1 is distinguishable from Leach et al. and Ouellette. Accordingly, claims 6 and 12 are also distinguishable from these references for the same reasons set forth in connection with claim 1, and Appellants respectfully request that the Board reverse the rejection of claims 6 and 12.

Claims 2 and 7 depend from claims 1 and 6, respectively. As explained, claims 1 and 6 are distinguishable from Leach et al. and Ouellette. Accordingly, claims 2 and 7 are also distinguishable from these references for the same reasons set forth in connection with claims 1 and 6, and Appellants respectfully request that the Board reverse the rejection of claims 2 and 7.

d. The Examiner's position that Leach et al. teaches the recitations of claims 4 and 9 is wrong because determining which interface to retrieve and how to invoke the interface is not the same as the specifying steps recited in claims 4 and 9, and the query function member taught by Leach et al. is not an object and there is no evidence in the reference that the function member is specified when a multitype object is created.

In support of the rejection of claims 4 and 9 under 35 U.S.C. § 103(a), the Examiner asserts that Leach et al. teaches the step of specifying an object, which is recited in claims 4 and 9, at col. 23, lines 9-10 (see Office Action dated October 7, 2003, page 4, lines 5-9). Appellants respectfully disagree for the following reasons.

First, the process of determining which interface to retrieve and how to invoke the interface as taught by Leach et al. is not the same as the specifying step recited in claims 4 and 9. As explained in the October 7, 2002 response, Leach et al. provides a method for modifying a determination of which interfaces to retrieve and how to invoke them "in combination if more than one instance of the same interface is present in the aggregate object" (i.e., enclosing object) (see Leach et al., col. 23, lines 9-12). This process taught by Leach et al. is not equivalent to specifying an object to process method invocations on the instance, as recited in claims 4 and 9 because, at the very least, there is no specification of an object that processes method invocations. Moreover, the method provided by Leach et al. is limited to information on how to invoke interfaces in combination **if more than one instance of the same interface** is present. This limitation has no relationship to the recitations of claims 4 and 9.

The Examiner responds to these arguments in the October 7, 2003 Office Action by simply reiterating the position set forth in the Final Office Action, *i.e.*, that the disclosure in col. 23, lines 9-10 of Leach et al. teaches the specifying step of claim 4. Accordingly, the Examiner has yet to present evidence that Leach et al. teaches the recitations of claims 4 and 9 in light of Appellants' arguments included in the Appeal Brief filed July 14, 2003, and the responses filed October 7, 2002 and February 11, 2003. Because the Examiner has not shown that Leach et al., or Ouellette, teach or suggest the recitations of claims 4 and 9, Appellants submit that the rejection of these claims under 35 U.S.C. § 103(a) is improper and should be reversed.

Second, the query function member that is invoked by Leach et al. merely retrieves a reference to an aggregated interface, and thus cannot teach the step of specifying an object to process method invocations on the instance, as recited in claim 4. The Examiner appears to take the position that the query function member of the multitype object taught by Leach et al. is the same as the object specified in claims 4 and 9. Appellants disagree on this point as well. The query function member is not an object and there is no evidence in Leach et al. that shows the function member is even specified when the multitype object is created. Therefore, the invocation of a query function member of the multitype object taught by Leach et al. is not the same as the specifying step of claims 4 and 9.

Because Leach et al. and Ouellette, alone or in combination, fail to teach or suggest every recitation of claims 4 and 9, Appellants respectfully request that the Board reverse the rejection of these claims.

e. The Examiner's position that Leach et al., Ouellette, and Hailpern et al. collectively suggest the recitations of claims 5 and 10 is wrong because these references fail to teach or suggest a handler that is in the same context as the invocation handler of these claims, there is no reasonable expectation of success in the combination of these references, and the class definitions described in table 3 of Leach et al. do not show generating at runtime a class that implements an interface, as recited in the claims.

Appellants traverse the rejection of claims 5 and 10 under 35 U.S.C. § 103(a) because a prima facie case of obviousness has not been made by the Examiner. As explained, to establish a prima facie case of obviousness under 35 U.S.C. § 103(a), the

reference or references, taken alone or combined, must teach or suggest each and every element recited in the claims; there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to combine the references in a manner resulting in the claimed invention; and a reasonable expectation of success must exist. (see M.P.E.P. § 2143 (8th ed. 2001)).

In rejecting claims 5 and 10, the Examiner admits that Leach et al. does not teach or suggest an invocation handler (see Final Office Action, page 4, paragraph 4). In an effort to supplement this missing element, the Examiner asserts that the process execution handler taught by Hailpern et al. is equivalent to an invocation handler and that it would have been obvious to apply the execution handler to the system of Leach et al. "because this allows dynamic aggregating objects" (see Office Action dated October 7, 2003, page 5, lines 2-4).

In the responses filed October 7, 2002 and February 11, 2003, Appellants presented arguments distinguishing claims 5 and 10 from Leach et al. and Hailpern et al. and explained why the Examiner failed to establish a prima facie case of obviousness. In response, the Examiner addressed one of the arguments presented by Appellants by merely stating that "while this may be true [the handler of Hailpern et al. is not in the same context as the handler claimed] it does not preclude using Hailpern in the claim rejections." The Examiner states nothing else regarding the issues involved with the rejections of claims 5 and 10 under 35 U.S.C. § 103(a).

Not only does the Examiner's position demonstrate an admission that the handler taught by Hailpern et al. is not analogous to Appellants claimed handler and Leach et al.'s object aggregation environment, but the Examiner is also wrong because the fact that Hailpern et al. does not teach or suggest (as Admitted by the Examiner) a handler that is in the same context as the invocation handler of claims 5 and 10 does preclude using the reference to reject these claims under 35 U.S.C. § 103(a).

As explained above, the Examiner has a burden to show (1) that Hailpern et al. and Leach et al., taken alone or combined, teaches or suggests each and every element recited claims 5 and 10 (see M.P.E.P. § 2143.03 (8th ed. 2001)), (2) that there is some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to combine the Hailpern et al. and Leach et al. in a manner resulting in the inventions recited in claims 5 and 10, and (3) that there is a reasonable expectation of success in making the combination.

The Examiner has failed to meet this burden and in fact has failed to make any of these required showings in connection with the rejection of claims 5 and 10. As explained by Appellants in previous responses, the process execution handler taught by Hailpern et al. carries out "necessary processing" for a server that performs virus checking in a server network environment (see Hailpern et al., col. 15, lines 48-40). The execution handler is not an invocation handler that receives method invocations, as recited in claims 5 and 10.

Also, there is no reasonable expectation of success in implementing a virus scanner exception handler, as taught by Hailpern et al., in the method invocation

processes taught by Leach et al. One of ordinary skill in the art would have appreciated, at the time of the invention, that such a combination is not suggested in the references themselves in any possible interpretation and that the asserted combination is not feasible because the virus scanner taught by Hailpern et al. does not perform in a manner consistent with the handler recited in claims 5 and 10. If anything, the combination asserted by the Examiner merely results in Leach et al. using virus scanning processes, which is not analogous to Appellants' claimed invention. Further, there is no suggestion in either reference that would motivate one of ordinary skill in the art to use virus exception handlers in the system for aggregating objects taught by Leach et al.

Moreover, when given the opportunity to elaborate on the rejection of claims 5 and 10, the Examiner merely concluded that Hailpern et al. may be used to reject these claims without presenting evidence to support the conclusion (see Final Office Action, page 6, lines 11-13). Indeed, the Examiner relies on the same rejection set forth in the Final Office Action. (See Office Action dated October 7, 2003, page 4, paragraph 4). Accordingly, Appellants submit the Examiner's position is improper and requests that the Board reverse rejection of claims 5 and 10.

The Examiner also maintained the position that Leach et al. teaches generating at runtime a class that implements the interface by generating code for each of the methods included in the interface, as recited in claims 5 and 10 (see Final Office Action, page 6, lines 14-21). The Examiner asserts that Leach et al. discloses this recitation in Code table 3 and columns 11-14.

The Examiner is wrong. The code table 3 taught by Leach et al. is a listing of pseudo code for **class and method definitions of an object** enclosed in an aggregate object (see Leach et al., col. 14, lines 23-28). There is no mention or suggestion in Leach et al., particularly associated with code table 3, of generating a class at runtime that implements an interface by generating code for each of the methods in the interface, as recited in claims 5 and 10. Further, merely presenting a class definition does not show **generating at runtime** a class that implements an interface by generating code for each of a plurality of methods of an interface indicated at runtime. Also, the objects created dynamically at run time, as disclosed in column 11, lines 22-23 of Leach et al. (cited by the Examiner) are instances of **statically aggregated objects**, and not a class that implements an interface, as recited in claims 5 and 10. Accordingly, Leach et al. cannot teach or suggest the generating step recited in these claims.

Because Leach et al. fails to teach or suggest the recitations of claims 5 and 10, and Ouellette and Hailpern et al. fail to make up for the deficiencies of Leach et al., Appellants request that the Board reverse the rejection of these claims.

f. The Examiner's position that Leach et al., Ouellette, Hailpern et al., and Hughes collectively suggest the recitations of claims 11 and 13 is wrong because these references fail to teach or suggest a handler object and a proxy class as recited in these claims, returning a value from the object to a proxy class instance, as recited in claim 11, and a processor for generating a proxy class at runtime, receiving a request, and dispatching a request, as recited in claim 13.

In rejecting claims 11 and 13 under 35 U.S.C. § 103(a), the Examiner relies on the rejections of claims 1, 2 and 5 (referring to the teachings of Leach et al., Ouellette, and Hailpern et al.). Further, the Examiner relies on Hughes to teach a proxy class, as recited in these claims. Appellants disagree with the Examiner's position for the following reasons.

Leach et al., Ouellette, and Hailpern et al., alone or in combination, do not teach or suggest a handler that is in the same context as the handler object of claims 11 and 13. As explained above with reference to the rejection of claims 5 and 10, the process execution handler taught by Hailpern et al. carries out "necessary processing" for a server that performs virus checking in a server network environment (see Hailpern et al., col. 15, lines 48-40). The execution handler is not a handler object, as recited in claims 11 and 13. Further, as explained, the code table 3 disclosed by Leach et al. does not teach generating at runtime a class implementing a determined interface. Claim 11 is also distinguishable from Hughes because that reference does not teach or suggest, alone or in combination with Leach et al., Ouellette, and Hailpern et al., at least dispatching a request to an invocation handler object and returning a value from the handler object to a proxy class instance, as recited in the claim.

Further, the Examiner maintains the position that Hughes teaches a proxy class generated at runtime, as recited in claims 11 and 13. The Examiner reaches this conclusion in the Office Action by reiterating the arguments presented in the Final Office Action dated December 18, 2002 and the Office Action dated July 17, 2002 without providing evidence or an explanation to support the conclusion in light of

Appellants arguments presented in the Appeal Brief filed July 14, 2003, and the responses filed October 7, 2002 and February 11, 2003 (see Office Action dated October 7, 2003, page 5, paragraph 5, Final Office Action, page 6, lines 1-10 and page 7, lines 1-4, and the Advisory Action, page 2) . Accordingly, Appellants reiterate the arguments presented in the October 7, 2002 response and the Appeal Brief filed July 14, 2003 and respectfully request that the Examiner and the Board address them accordingly.

As explained by Appellants, the proxy class disclosed by Hughes is not generated at runtime. Instead, a manufacturer generates code for interfaces, and other functionalities associated with the proxy class (see Hughes, col. 4, lines 18-31 and 60-67). As disclosed by Hughes, a proxy class is not generated at runtime, but rather a **user dynamically specifies at runtime** the "customization" of an instance of the proxy class (see Hughes, col. 8, lines 6-8). Therefore, Hughes cannot teach a proxy class, as recited in claims 11 and 13.

Because Hughes, Leach et al., Ouellette, and Hailpern et al., alone or in combination, fail to teach or suggest the recitations of claims 11 and 13, Appellants respectfully request that the Board reverse the rejection of these claims.

IX. CONCLUSION

The rejection of claims 1, 2, 4- 7, and 9-13 should be reversed because Leach et al., Ouellette, Hailpern et al., and/or Hughes, alone or in combination, do not teach or suggest every recitation of these claims and because the Examiner has not established


a prima facie case of obviousness in rejecting these claims under 35 U.S.C. § 103(a). Accordingly, Appellants respectfully request such reversals.

To the extent any extension of time under 37 C.F.R. § 1.136 is required to obtain entry of this Appeal Brief, such extension is hereby respectfully requested. If there are any fees due under 37 C.F.R. §§ 1.16 or 1.17 which are not enclosed herewith, including any fees required for an extension of time under 37 C.F.R. § 1.136, please charge such fees to our Deposit Account No. 06-0916.

Respectfully submitted,

FINNEGAN, HENDERSON, FARABOW,
GARRETT & DUNNER, L.L.P.

By: _____


Joseph E. Palys
Reg. No. 46,508

Dated: January 6, 2004

Post Office Address (to
which correspondence is
to be sent)

Finnegan, Henderson, Farabow,
Garrett & Dunner, L.L.P.
1300 I Street, N.W.
Washington, D.C. 20005
(202) 408-4000

FINNEGAN
HENDERSON
FARABOW
GARRETT &
DUNNER LLP

1300 I Street, NW
Washington, DC 20005
202.408.4000
Fax 202.408.4400
www.finnegan.com

APPENDIX

1. A method in a data processing system comprising the steps of:
generating at runtime a class that implements an interface specified at runtime
having a method;
creating an instance of the class;
receiving by the class instance a request to process the method of the interface;
dispatching the request to an object to facilitate processing of the method of the
interface; and
returning a result of the processed method by the object.
2. The method of claim 1, wherein the step of generating a class further includes
the step of:
generating at runtime a class that implements more than one interface specified
at runtime, each interface having one or more methods.
4. The method of claim 1, wherein the step of creating an instance further
includes the step of:
specifying an object to process method invocations on the instance.

5. A method in a data processing system having an invocation handler, comprising the steps of:

receiving at runtime an indication of at least one interface having a plurality of methods; and

generating at runtime a class that implements the interface by generating code for each of the methods that dispatches an invocation of the method to the invocation handler.

6. A computer-readable medium containing instructions for controlling a data processing system to perform a method comprising the steps of:

generating at runtime a class that implements an interface specified at runtime having a method;

creating an instance of the class;

receiving by the class instance a request to process the method of the interface;

dispatching the request to an object to facilitate processing of the method of the interface; and

returning a result of the processed method by the object.

7. The computer-readable medium of claim 6, wherein the step of generating a class further includes the step of:

generating at runtime a class that implements more than one interface specified at runtime, each interface having one or more methods.

9. The computer-readable medium of claim 6, wherein the step of creating an instance further includes the step of:

specifying an object to process method invocations on the instance.

10. A computer-readable medium containing instructions for controlling a data processing system having an invocation handler to perform a method comprising the steps of:

receiving at runtime an indication of at least one interface having a plurality of methods; and

generating at runtime a class that implements the interface by generating code for each of the methods that dispatches an invocation of the method to the invocation handler.

11. A method in a data-processing system having a proxy class implementing a list of interfaces specified at runtime, the interfaces containing methods, the system further having an instance of the proxy class and an invocation handler object for executing the methods contained by the interfaces, the method comprising the steps of:

determining interfaces to be implemented by the proxy class;

generating at runtime the proxy class implementing the determined interfaces;

receiving a request by the proxy class instance to invoke a method of an interface implemented by the proxy class;
dispatching the request to the invocation handler object;
returning a value from the invocation handler object to the proxy class instance;
and
returning the value from the proxy class instance.

12. A data-processing system comprising:

means for generating at runtime a class that implements an interface specified at runtime having a method;

means for creating an instance of the class;

means for receiving by the class instance a request to process the method of the interface;

means for dispatching the request to an object to facilitate processing of the method of the interface; and

means for returning a result of the processed method by the object.

13. A data-processing system comprising:

a memory containing a proxy class, an instance of the proxy class, an interface specified at runtime having methods, and an object for handling method invocations, the proxy class implementing the interface; and

a processor for generating the proxy class at runtime, receiving a request to access a method of the interface and dispatching the request to the object to facilitate processing of the requested method of the interface.